UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

EAI FRAMEWORK

Mário Filipe Santos Dias

Trabalho orientado pelo Prof. Doutor Dimitris Mostrous
e coorientado por José Miguel Leal e Silva

TRABALHO DE PROJETO
(Versão pública)

MESTRADO EM ENGENHARIA INFORMÁTICA
Arquitetura, Sistemas e Redes de Computadores

2016

Agradecimentos

Quero começar por agradecer a todos os professores que tive na vida, sem eles não saberia o que sei hoje. Aos meus amigos, que sempre tiveram lá, um agradecimento especial ao André Campos e um muito especial ao Rui Teixeira. Obrigado José Leal por teres acreditado em mim, e a toda a tua equipa da BEE ENG. de excelentes profissionais, nunca vos esquecerei, sem vocês seria mais um. João Martins e Aires Noronha dois dos melhores profissionais, cada um na sua área a nível nacional e internacional, obrigado por confiarem em mim tamanhas responsabilidades só confiando na minha palavra, e agradecer o convite de trabalhar com profissionais de tão alto gabarito, com projetos internacionais de elevada importância que muitos profissionais da minha área não terão a oportunidade de meter isso nos seus currículos. Trabalharei para não vos desiludir. Aos meus pais, um apenas "Já está, fim, esta luta está ganha, vamos para casa, venha o próximo capítulo". Agora é a minha vez de vos retribuir e de cuidar de vós, só espero vir a ser um pai tão bom como vocês foram. Mãe, respira fundo… tu sabes. Ao meu amor Andreia, que aturou algumas das minhas diretas, as minhas birras, e o meu mau feitio, muito do bem que me vai acontecendo é culpa tua e ainda bem. Não me achando presunçoso, quero agradecer ao meu mau feitio que nunca deixou que os meus medos e fraquezas levassem a melhor, fazendo com que me erguesse depois de lutas parecidas perdidas.

Um agradecimento num parágrafo à parte a uma pessoa que gerou a minha mãe, a quem muitos chamam avó, mas que eu considero santa. "A velha" será para sempre eterna, Chuck Norris que me defendia a mim e à minha mãe, durando quase o triplo da data de validade do seu coração. Eu sei que este trabalho é chinês para ti, mas com a tua ajuda consegui acabar o sonho que tinha desde pequeno, quando tu eras a minha companhia. Nunca te agradeci, desculpa, mas irei fazer-te a vontade: "Agradeço a Deus, Jesus, Nossa Senhora de Fátima, Nossa Senhora do Milagres". As lágrimas escorrem pelo rosto e não te tenho para mas limpares…

“Vó, Oh Vó, Vóóó, olha o que eu fiz para ti.”

Resumo

Este projeto tinha como objetivo o desenvolvimento de uma *Enterprise Application Integration* (EAI) *Framework* para integração dos vários sistemas do banco Co-operative Bank no Quénia. A *framework* e os serviços serão desenvolvidos utilizando *software* TIBCO, como TIBCO *ActiveMatrix BusinessWorks™ BPM*, TIBCO *Enterprise Message Service™* e TIBCO *Designer™*. Foram aplicados os conhecimentos de arquitetura *Service Oriented Architecture* (*SOA)*, para uma melhor e mais eficaz *framework*, em serviços a serem utilizados por sistemas heterogéneos. Os requisitos do banco eram:

- Os serviços podem ser síncronos ou assíncronos.
- Todos os serviços assíncronos terão de dar uma resposta com o resultado da operação.
- Uma manutenção do sistema reduzida e simples.
- Uma integração fácil entre os diferentes sistemas do banco.
- Baixo índice de *bugs* por serviço; e facilidade em descobri-los, analisá-los e corrigi-los.

A *framework* é responsável pelo *log* das atividades dos serviços na camada de integração, pela validação dos cabeçalhos, pelo encaminhamento das mensagens entre as duas camadas que compõem a *framework* (*Business* e *Technical*). Fica ainda responsável pela gestão das mensagens em serviços assíncronos, onde dará uma resposta genérica à origem, informando que a mensagem se encontra a ser processada pela *framework*, enviando posteriormente outra resposta genérica com o resultado da operação no destino.

Durante o desenvolvimento da *framework*, foi tida em atenção a normalização dos nomes, tanto para *target namespace* dos *schemas*, ficheiros, e outros itens que são transversais a toda a *framework*, bem como as regras para aumento das versões *major* e *minor*.

Dos vários serviços que utilizaram a *framework*, pode-se dar o exemplo de um serviço que te tem como origem *Customer Relationship Management* (CRM) e o destino *BankFusion Universal Banking* (BFUB). O serviço retornará os vários balanços de uma conta, dando como *input* o número de uma conta. O serviço é um serviço síncrono que o depois de 60 segundos o serviço retornará um erro de *timeout.*

Palavras-chave: TIBCO, *Framework, SOA, EAI, ESB*

Abstract

This project had the objective to develop an Enterprise Application Integration (EAI) Framework for the integration of the various systems in Co-operative Bank in Kenya. The framework and the services will be developed using TIBCO software, such as TIBCO ActiveMatrix BusinessWorks™ BPM, TIBCO Enterprise Message Service™ and TIBCO Designer™. We applied our expertise in *Service Oriented Architecture* (SOA) architecture, for a better and more effective framework, in those services to be used by heterogeneous systems. The bank's requirements were:

- Services can be synchronous or asynchronous.
- All asynchronous services will have to respond with the outcome of the operation.
- A reduced and simple system maintenance.
- An easy integration between different bank systems.
- A low "bug" rate per service; and an easiness in finding, analysing and rectifying it.

The framework is responsible for logging the service activities on the integration layer, through validation of headers, routing of messages between the two layers the make up the framework (Business and Technical). It will also be responsible for managing the messages in asynchronous services, which will send a generic reply to the origin, informing that the message is being processed by the framework, sending subsequently another generic reply with the result of the operation at the endpoint.

During the development of the framework, the normalization of names has been taken into account, for both target namespace of schemas, files and other items that are transversal to the entire framework, as well as the rules to increase the major and minor versions.

From the several services that will use the framework, it can give you an example of one service, that as *Customer Relationship Management* (CRM) from origin and *BankFusion Universal Banking* (BFUB) as destiny It will return the several balances of an account, giving in the input the account number. The service is a synchronous service, and after 60 seconds it will reply with a timeout error.

Keywords: TIBCO, *Framework*, *SOA*, *EAI*, *ESB*

Conteúdo

Lista de Figuras

Lista de Acrónimos

AIA	Application Integration Architecture
BFUB	Fusion Banking Essence
BW	TIBCO Business Works
CRM	Customer Relationship Management
EAI	Enterprise Application Integration
EAR	Enterprise Archive
EMS	TIBCO Enterprise Message Service
ESB	Enterprise Service BUS
HTTP	Hyper Text Transfer Protocol
JMS	Java Messaging Service
QA	Ambiente de Qualidade
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
WSDL	Web Services Description Language
XML	Extensible Markup Language

Capítulo 1

Introdução

Num mundo onde as empresas têm de se atualizar constantemente para serem competitivas, e em que o negócio ou os produtos que disponibilizam também mudam constantemente, ter o negócio alinhado com as tecnologias de informação é fundamental para ter uma empresa de sucesso. As tecnologias de informação têm de ser dinâmicas e adaptarem-se o mais rápido possível ao negócio, mas devido à grande dimensão da empresa ou do negócio, e o facto de as empresas conterem vários negócios em paralelo ou até mesmo a junção das características atrás referidas, esse mesmo alinhamento entre tecnologias de informação e negócio torna-se complexo. Essa complexidade de alinhamento entre os dois é o resultado da heterogeneidade dos sistemas dentro e fora da empresa. Muitos sistemas encontram-se fora da empresa, sem controlo por parte desta. E mesmo dentro da empresa vários sistemas têm modos diferentes de comunicação e de processamento interno da informação. Aliando a isso o paradigma de comunicação ponto-a-ponto, percebemos que podemos facilmente perder o controlo e/ou visão do sistema informático da empresa. O paradigma de comunicação ponto-a-ponto aumenta a complexidade, tempo e custos de desenvolvimento de cada sistema. O que leva que o alinhamento com o negócio, não se torne dinâmico.

É por estes motivos que a integração é tão importante. A integração facilita esse alinhamento entre as tecnologias de informação e o negócio, baixando os custos e a complexidade dessa mesma operação. Aliando a integração com uma *framework* de serviços, esta irá reduzir ainda mais os custos e a complexidade de desenvolvimento de serviços que iram integrar os vários sistemas da empresa.

1.1 Motivação

Este projeto surge no âmbito de um projeto que me foi apresentado pela Bee Engineering, desafio que aceitei de imediato por várias razões. Uma das razões é que já contava com alguns anos de experiência com este tipo de *software* e paradigma. Outra das razões prende-se com a importância que a integração desempenha hoje em dia, sendo um dos ramos mais críticos e importantes das tecnologias de informação de uma empresa [1].O projeto era integrar todos os sistemas do Co-operative Bank of Kenya, um conceituado banco no Quénia.

Os sistemas do Co-operative Bank of Kenya não estavam integrados, sendo que todo o paradigma de comunicação assentava na conexão ponto-a-ponto. Com a conexão ponto-a-ponto, cada consumidor terá conexões diferentes a um único fornecedor para o mesmo serviço ou recurso. Com o paradigma de conexão ponto-a-ponto, a adição de novos sistemas, serviços e recursos, que ao longo do tempo é preciso sempre acrescentar mais. Esse acrescentar vai ficando cada vez mais complexo, com maiores custos de

Figura 1 Exemplo de conexão ponto-a-ponto entre várias aplicações e sistemas

desenvolvimento e de manutenção.

Como podemos ver na *Figura 1*, um exemplo de conexões de ponto-a-ponto entre várias aplicações e sistemas. Não é preciso ter muitos serviços ou sistemas para que a administração e manutenção dos sistemas se tornem complexas. A solução passa por integrar todos os sistemas, o que aconteceu neste projeto com *software* da TIBCO e utilizando SOA como estilo de arquitetura, a mais adequada para este tipo de integrações.

1.2 Co-operative Bank of Kenya

O Co-operative Bank of Kenya é o cliente onde esta *framework* foi propositadamente desenvolvida e instalada, para que se pudesse integrar os sistemas do

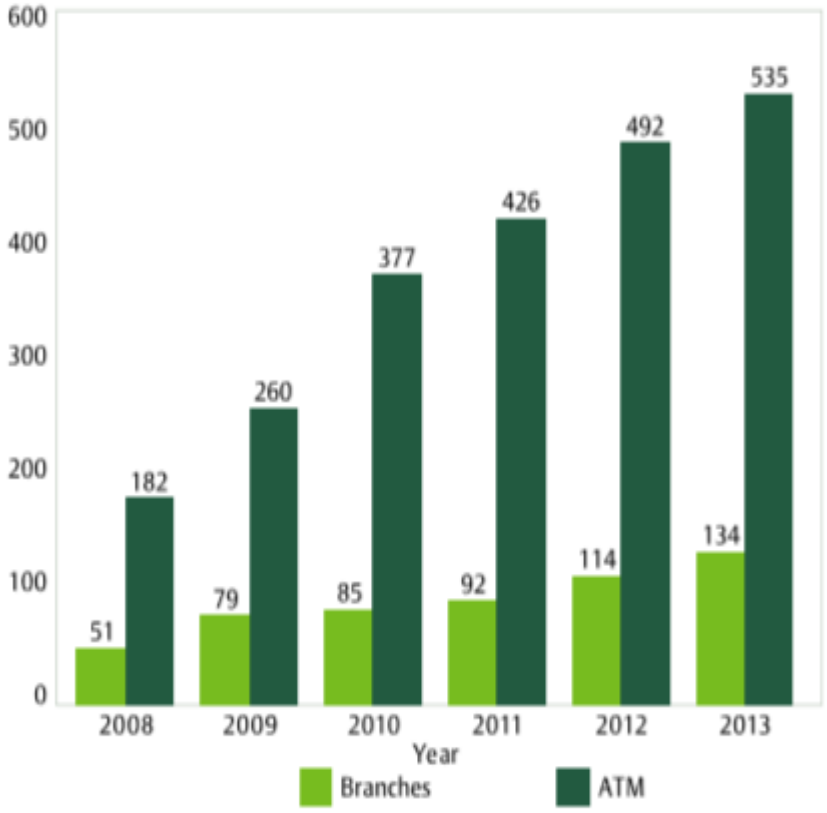

Figura 2 Em verde-claro o número de sucursais bancárias e a verde-escuro o número de ATM *possuídos pelo banco.*

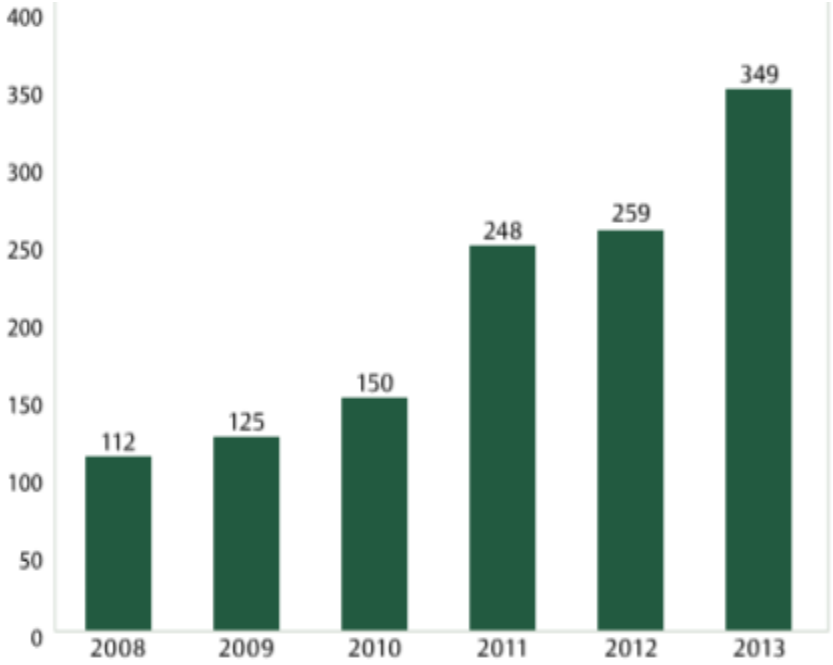

Figura 3 Número de clientes em milhares ao longo dos anos.

banco. O Co-operative Bank of Kenya foi, em 2013, o terceiro banco do Quénia em termos de lucro[1].

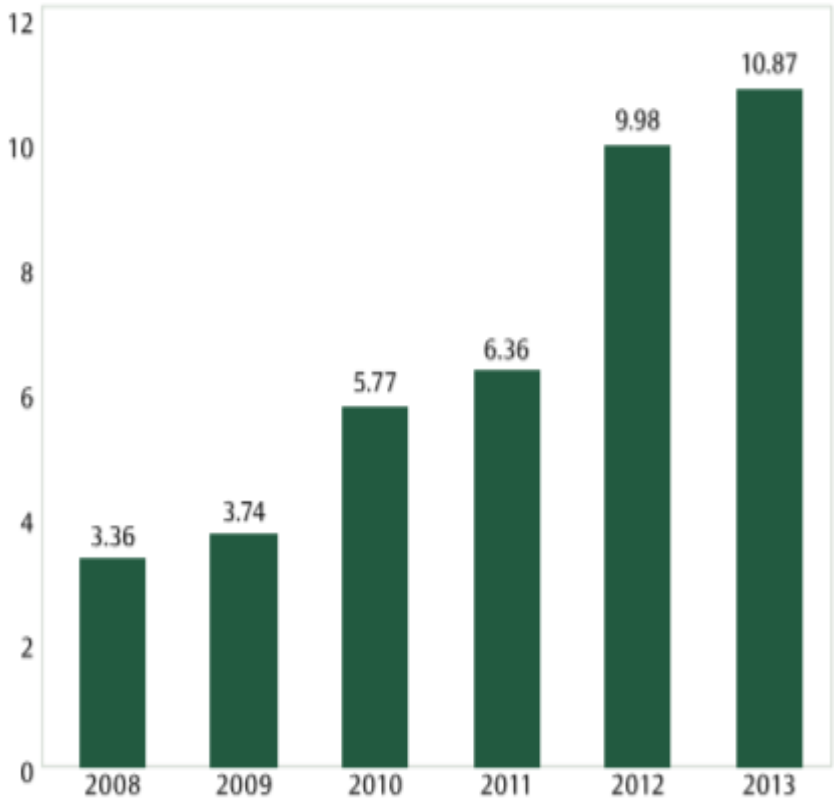

Figura 4 Lucro em mil milhões de shillings *quenianos por ano*

Em 2013, obteve um lucro bruto de 10,87 mil milhões de *shillings* quenianos e, com 134 sucursais bancárias, é considerado um banco de grande dimensão no Quénia [2]. Como podemos ver na Figura 2, 3 e 4 é um banco que ao longo dos anos tem estado a crescer tanto nos lucros como no número de clientes e agências.

Uma vez que o banco procura reduzir os custos e aumentar a eficácias dos serviços que disponibiliza, decidiu começar a integrar serviços antigos que utilizavam o paradigma de comunicação ponto-a-ponto, e novos serviços que seriam desenvolvidos já com o paradigma SOA.

O banco contém atualmente 12 sistemas que serão integrados:

- BFUB - *BankFusion Universal Banking* é o sistema *core* do banco.
- CIB - *Corporate Internet Banking* é o *website* bancário para empresas.
- CMS - *Card Management System* é o sistema de gestão de cartões de crédito e débito do banco.

[1] Http://bankelele.co.ke/2014/04/kenya-bank-rankings-2013-part-i.html

- CRM - *Customer Relationship Management* é o sistema que tem a responsabilidade de gestão de relacionamento com o cliente.
- M-Wallet - *Mobile Wallet* é uma plataforma para pagamento com telemóveis.
- M-Wallet Gateway - é o sistema responsável pelo envio de SMS.
- PCS - é o sistema de guarda de pensões.
- RIB - *Retail Internet Banking* é o *website* bancário para clientes de retalho.
- RTPS - *Real-Time Publish Subscribe* é onde estão guardadas todas as transações que não sejam espoletadas por BFUB.
- SMTP – sistema que possui o mesmo nome do protocolo de SMTP, que permite o envio de correio eletrónico.
- SYBRIN – reconciliação de mensagens *swift*.

Como se viu anteriormente não se trata de um cliente pequeno, ou de um banco pequeno, por isso a importância dada à *framework* é elevada. Caso algo falhe na *framework*, o número de cliente é bastante elevada, bem como os prejuízos resultantes dessa mesma falha. Por estas razões, foi necessário ter enorme atenção até ao mais pequeno detalhe.

1.3 *Service-Oriented Architecture*

Service-oriented architecture ou SOA, que traduzido para português é Arquitetura Orientada a Serviços, é um modelo de desenho de *software* em que todas as aplicações disponibilizam serviços, para que outras aplicações possam invocar esses mesmos serviços. Um serviço é uma representação lógica de uma atividade de negócio repetitiva e que tem um resultado específico (como por exemplo, consular o balanço de uma conta de um cliente). O serviço é uma "caixa negra" para os consumidores do serviço [3]. Ou seja os consumidores não sabem nada de como o serviço funciona, qual a sua lógica ou que linguagem de programação utiliza para implementar essa lógica. Tudo o que o consumidor sabe é o *input* e *output* do serviço

Um serviço é desenhado para realizar uma atividade, implementando uma ou mais operações. Como resultado, cada serviço tem código e lógica independente do outro. Assim, é possível reutilizar os serviços, mesmo que se tenha de fazer alguma alteração, esta é feita nesse determinado serviço e não no sistema que faz o pedido, ou no sistema final, o que seria mais complexo, devido a possíveis dependências. Os princípios de

desenho SOA são usados durante o tempo de desenvolvimento e implementação. SOA em vez de definir uma API, define a interface em termos de protocolos e funcionalidades.

Não existem padrões exatos para a composição de uma Arquitetura Orientada a Serviços, embora muitas empresas que são as criadoras de ferramentas para integração tenham publicado os seus princípios. Alguns dos princípios publicados [4,5,6] incluem os seguintes pontos:

- Contrato de serviço padronizado: Os serviços aderem a um acordo de comunicações, como foi definido coletivamente por um ou mais documentos de descrição de serviços.
- Serviços de baixo acoplamento: Os serviços mantêm uma relação que minimiza dependências, apenas exigindo que tenham consciência uns dos outros.
- Abstração do serviço: Para além da descrição do contrato de serviço, o serviço esconde a sua lógica do mundo exterior.
- Reutilização de serviços: A lógica é dividida em serviços com a intenção de promover a reutilização.
- Autonomia do serviço: Os serviços têm o controlo sobre a lógica que encapsula, numa perspetiva de tempo de desenho e também de tempo de execução.
- Serviços sem estado: Minimizam o consumo de recursos ao eliminar a necessidade de gestão de informação de estado.
- Facilidade de descoberta de serviços: Os serviços são complementados com metadados, com os quais serão mais facilmente descobertos e interpretados.
- Granularidade do serviço: Uma consideração em tempo de desenho é providenciar um âmbito ideal e um nível adequado de granularidade da funcionalidade de negócio numa operação de serviço.
- Normalização de serviço: Serviços decompostos e/ou consolidados até certo nível, de uma forma normal, para minimizar a redundância. Em alguns casos, os serviços são desnormalizados para propósitos específicos, tais como otimização de performance, acessos, e de agregação [7].

- Otimização de serviços: Se tudo o resto for igual, os serviços de melhor qualidade são preferíveis aos de menor qualidade (por razões óbvias).
- Relevância dos serviços: Funcionalidades são apresentadas numa granularidade reconhecida pelo utilizador como um serviço relevante.
- Encapsulação de serviços: Muitos serviços são consolidados para uso com SOA. Muito desses mesmos serviços não foram planeados para serem usados com SOA.
- Transparência na localização de serviços: Refere-se à capacidade de um consumidor de um serviço de invocar o serviço independentemente da sua localização na rede. Isto também é reconhecido como Facilidade de descoberta de serviços (um dos princípios fundamentais de SOA) e o direito de um consumidor ter acesso ao serviço. Muitas vezes, a ideia de virtualização de serviços remete para a transparência na localização de serviços. Isto é, quando o consumidor simplesmente invoca um serviço lógico, um componente em tempo de execução na infraestrutura SOA, normalmente um "*service bus*", mapeia esta invocação lógica para um serviço físico.

Uma parte muito importante na implementação de uma arquitetura orientada a serviços são os "*web services*" [8]. Os "*web services*" são como blocos de construção que podem ser acessíveis através de protocolos de Internet padrão e independentes de plataformas e de linguagens de programação. Estes serviços podem representar novas aplicações ou serviços para sistemas já considerados "ultrapassados", tornando mais adequada a comunicação com outros sistemas mais modernos. Cada "*web service*" pode ser dividido por fornecedor ou por consumidor. Os fornecedores disponibilizam serviços e fornecem um WSDL (*Web Services Description Language*), que descreve o serviço que prestam. WSDL é uma especificação do que os pedidos podem fazer, com que parâmetros, e especifica o que o serviço envia na resposta. É uma especificação completa de uma API. Os consumidores ou clientes consomem esse mesmo WSDL, para que consigam fazer chamadas ao servidor com sucesso.

Alguns arquitetos corporativos acreditam que SOA ajuda o negócio a responder com maior eficácia e rapidez às alterações no mercado [9]. Este tipo de arquitetura promove a reutilização de serviços, o que leva a muito menos desenvolvimento, sendo que qualquer alteração feita a um serviço existente terá uma menor complexidade do que fazer um de raiz. A ideia é que com SOA as organizações olhem para o problema de uma forma holística. Só sabendo a composição do negócio das organizações é que se poderá fazer

uma boa arquitetura. SOA consegue encapsular infraestruturas orientadas ao negócio, o que leva a uma construção mais lógica dos serviços, bem como a uma manutenção e gestão mais fáceis. Não só a manutenção e gestão são mais fáceis, assim como a visualização da arquitetura por pessoas ligadas ao negócio, ou por outras partes interessadas que não sejam técnicas em informática. É o negócio que molda os serviços e não os serviços que moldam o negócio.

Para a implementação da arquitetura SOA, irei utilizar um padrão utilizado por *web servers* para a parte da comunicação, que utilizará SOAP. O padrão SOAP usa internamente XML para enviar e receber dados dos serviços. As mensagens SOAP têm uma estrutura rígida em XML que ao serem recebidas, é preciso fazer um *parse* a essa mesma mensagem.

1.4 Enterprise Service Bus

O *Enterprise Service Bus* ou ESB é um modelo de arquitetura de *software* usado para desenho e implementação de comunicações entre vários sistemas, utilizando uma arquitetura SOA. Primariamente, é utilizado numa *Enterprise Application Integration* (EAI) para integração de sistemas e aplicações heterogéneas. Como se pode ver o

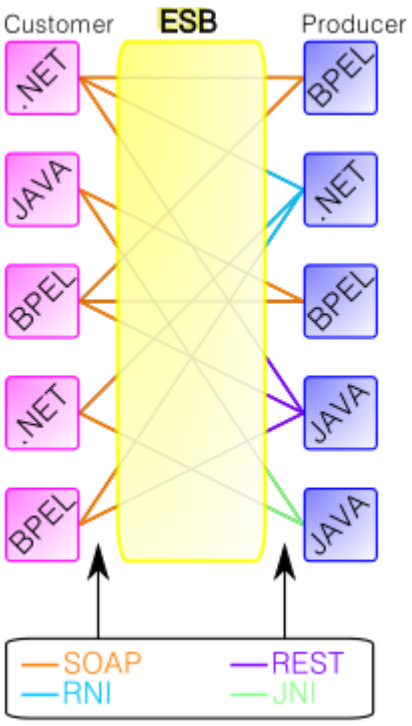

Figura 5 Todos os consumidores comunicam com o ESB da mesma maneira. O ESB traduz as mensagens para cada fornecedor específico

exemplo na Figura 5, o ESB normaliza todas as comunicações de entrada que depois traduz para protocolos específicos dos sistemas de destino.

Os deveres de um ESB são [10]:

- Monitorizar e controlar o encaminhamento das mensagens que são trocadas entre os serviços.
- Resolver os conflitos de comunicação entre os componentes dos serviços.
- Controlo de instalação e de versões de serviços.
- Uso imperativo de serviços redundantes.
- Providenciar serviços comuns, como tratamento de eventos, transformação e mapeamento de dados, serviço de fila de mensagens e eventos, segurança e tratamento de exceções, conversão de protocolos e impor uma qualidade de comunicação de serviços.

Os padrões de desenho de um ESB, [10,17] podem ser classificados nas seguintes categorias:

- Padrão de Interação: Ativar pontos de interação para envio e/ou receção a partir do barramento (*bus*).
- Padrão de Mediação: Ativação de alteração de troca de mensagens.
- Padrão de Implementação: Integração de uma solução de suporte na infraestrutura do ESB.

Exemplos de padrão de interação:

- Mensagem só de um sentido.
- Interação síncrona.
- Interação assíncrona.
- Interação assíncrona com tempo limite.
- Interação assíncrona com notificação por temporizador.
- Um pedido, múltiplas respostas.
- Um pedido, uma ou duas possíveis respostas.
- Um pedido, uma resposta obrigatória, e uma resposta opcional.
- Processamentos parciais.

Benefícios de um padrão de mediação para interação múltipla entre aplicações:

- O mediador promove o acoplamento fraco, mantendo os objetos a referirem-se um ao outro explicitamente, e permite várias interações, independentemente.
- Projetar um intermediário para desassociar muitos pares.
- Promover relações "muitos para muitos", entre pares interativos com "*full object status*" [11].

Exemplos de padrões de interações:

- ESB global: Os serviços partilham um único *namespace* e todos os prestadores de serviços estão visíveis a todos os solicitadores de serviços em toda a rede.
- ESB diretamente conectado: O registo de serviços globais que permitem a instalação no ESB está sempre visível.

1.5 TIBCO

Este projeto utiliza principalmente *software* da TIBCO para a *framework* e serviços. TIBCO é um acrónimo para *The Information Bus Company*, líder global de infraestruturas e *software Business Intelligence* [12]. A *ActiveMatrix BusinessWorks*™ é a plataforma da TIBCO para integração, e foi usada neste projeto. É considerada a plataforma de integração mais eficiente entre todas as plataformas de integração [12]. A plataforma já atrás referida é formada por várias ferramentas e *software*, e neste projeto foram utilizados os seguintes:

- TIBCO *Designer*: IDE de desenvolvimento com programação gráfica, com todo o código a ser gerado pelo IDE, sem que seja visível pelo utilizador.
- Cabe apenas ao utilizador adicionar os processos já previamente oferecidos pelo IDE ou criados à medida pelo utilizador. O utilizador só terá de mapear e/ou configurar os processos. Para mapear os campos de entrada ou de saída, poderá mapeá-los simplesmente ou utilizar XPath para mapeamentos com algum nível de lógica. Na Figura 6 podemos ver um exemplo de um processo desenvolvido no *Designer*.
- EMS: Servidor de mensagem, tanto por mensagens *Rendezvous* - protocolo proprietário da TIBCO -, como para mensagens JMS (*Java Message Service*).
- TIBCO *Run Time Agent*: Responsável por correr o código que foi criado no TIBCO *Designer*.
- TIBCO *Administrator*: Responsável pela administração/disponibilização *(deploy)* dos serviços na máquina; a interface é apresentada no navegador.

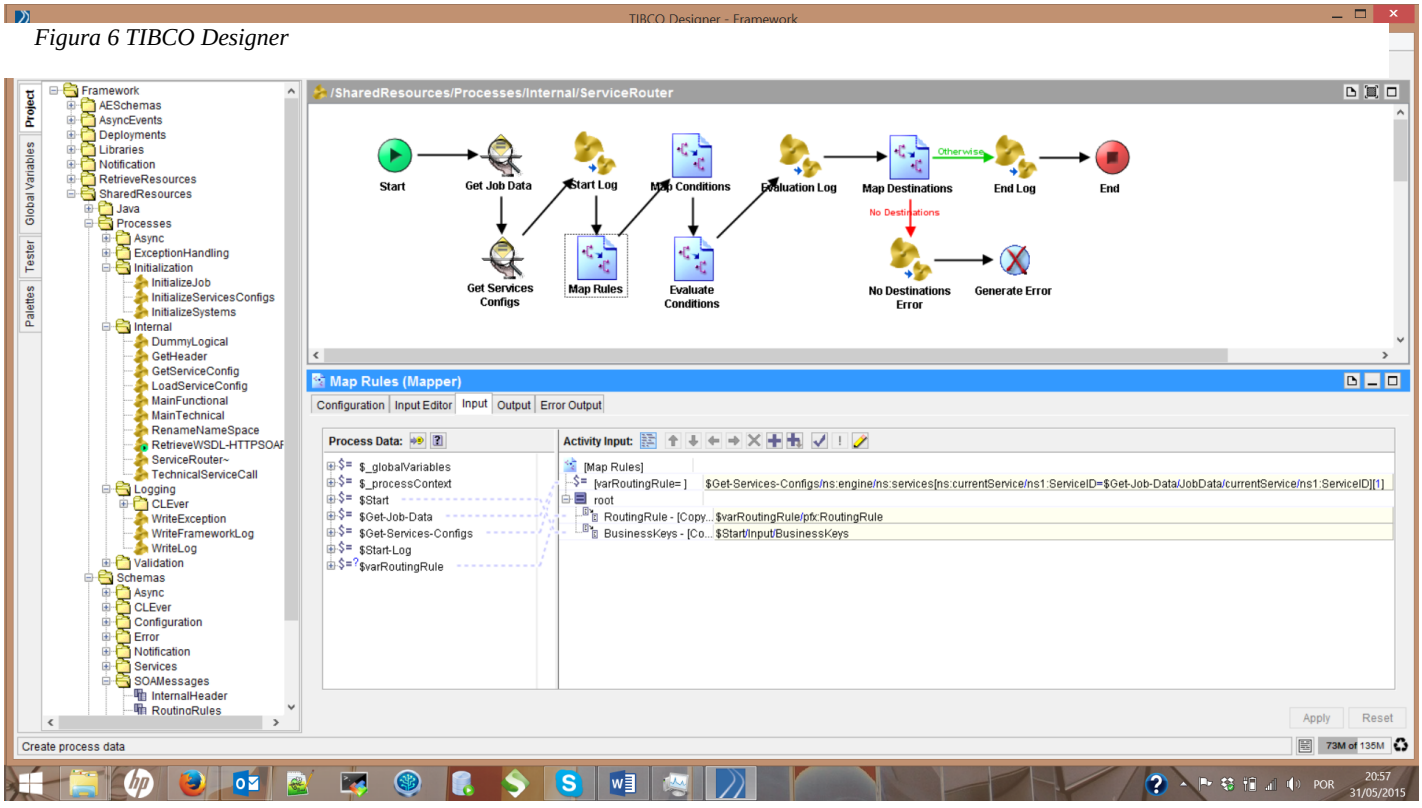

Figura 6 TIBCO Designer

1.6 Objetivos

O objetivo do projeto era a criação de uma *framework* para produzir um ESB para integração dos vários sistemas do banco. *Framework* na qual assenta todo o desenvolvimento e posterior processamento dos serviços. Também é possível criar um ESB sem *framework*, mas não será tão eficiente e/ou dinâmica com um ESB com *framework*. A *framework* teve de preencher todos os requisitos pedidos pelo cliente:

- Os serviços podem ser síncronos ou assíncronos.
- Todos os serviços assíncronos terão de dar uma resposta com o resultado da operação.
- Tolerante a faltas (tratamento de todo o tipo de erros, e recuperação de mensagens para serviços assíncronos).
- Uma manutenção do sistema reduzida e simples.
- Uma integração fácil entre os diferentes sistemas do banco.
- Baixo índice de *bugs* por serviço; e facilidade em descobri-los, analisá-los e corrigi-los.

Alem destes requisitos, foi tido em conta outro requisito pelo *Project Sponsor*: A *framework* teria de ser dinâmica, ao ponto de ser utilizada em mais projetos similares com o menor número possível de alterações. O *software* para o ESB teria de ser da TIBCO, o qual seria instalado em servidores dedicados, de forma a poderem existir 3 ambientes: Produção; Qualidade; Desenvolvimento.

1.7 Trabalho relacionado

No que diz respeito a *frameworks* em TIBCO, só conheço duas. Tive conhecimentos destas duas *frameworks* porque trabalhei em duas empresas que as utilizavam. Na Portugal Telecom (PT), tinham uma *framework* customizada baseada na *framework* genérica que a TIBCO instala nos seus clientes. Na TAP, que foi a outra a empresa em que trabalhei com um projeto TIBCO, a *framework* era a genérica que a TIBCO instala. Esta *framework* que foi produzida para este projecto, teve por base todos os conhecimentos que adquiri na PT.

Na PT existiam modelos para a construção dos serviços, mas não existia um repositório para *schemas* canónicos. A base de dados para a configuração dos serviços era dispersa, estando repartida por 3 bases de dados distintas. As respostas genéricas espoletadas pela *framework* não permitiam a emissão de informação para além do resultado do estado do processamento no sistema destino. A *framework* genérica da

TIBCO é uma *framework* muito básica, que pode ser considerada um ponto de partida para uma *framework* mais complexa.

Apesar das fragilidades da *framework* da PT, que eu os eliminei ao mesmo tempo que a fiz evoluir neste projeto. Na minha opinião, a *framework* da PT é umas das melhores *frameworks* que qualquer empresa poderá ter para os seus serviços. Apesar de não ser emocionante e algo monótona, desenvolver serviços para este tipo de *frameworks*, em termos empresariais é bastante eficaz. As equipas de desenvolvimentos são pequenas e muito eficazes, o que levava a uma enorme rentabilidade. Só por um exemplo, eu fiz parte da equipa de desenvolvimento que implementou o MEO4O da PT, em que a equipa consistia em apenas 3 membros que trabalharam durante um mês e meio. Este foi um dos maiores projetos até à atualidade dentro da PT, em que os serviços que tiveram que ser alterados ou feito de novo abrangiam todos os sistemas que a PT possuía na altura. Foi um enorme sucesso, só 3 pessoas da área de integração terem feito esse projeto colossal em tão pouco tempo, e sem a *framework* que a PT tinha na altura não teria sido possível tal projeto.

Ambas as *frameworks* (PT e da TAP) têm por base o mesmo conceito que a TIBCO promove junto dos seus clientes: a divisão do fluxo em dois principais agregadores de projetos TIBCO. A primeira parte contém todos os projetos que irão receber as mensagens dos serviços. Cada projeto agrupa todos os serviços que correspondam a uma parte do negócio, como: *Card*; *Customer*; *Account*, etc. A segunda parte agrupa todos os projetos que comunicam com os fornecedores. Cada projeto agrupa todos os serviços de um determinado fornecedor.

Cada empresa pode implementar o ESB e a sua *framework*, pelo que se torna difícil conhecer ou estudar todo o trabalho que foi desenvolvido nesta área, devido ao crescente número de clientes que a TIBCO possui.

Em relação a outros produtos no mercado para integração, que tenha tido contacto e algum conhecimento, são 2 produtos: *Application Integration Architecture* (AIA) da Oracle [13], e WSO2 *Enterprise Service Bus* [14], este último é 100% *open source*.

AIA tem como IDE o JDeveloper com alguns *bugs*, encontra-se numa fase de maturação. Vem já com *schemas* canónicos próprios para a indústria onde se destina a integração, telecomunicações, banca, etc. Estes *schemas* canónicos podem ser estendidos por *schemas* customizados pelo cliente. O preço é o maior aliado desta ferramenta, em que as licenças são menores do que as Licenças da TIBCO, que por vezes podem ser oferecida se o cliente comprar 2 ou mais produtos da Oracle, ou se já os tiver na sua empresa. O desenvolvimento com esta ferramenta é muito mais demorada do que as

ferramentas da TIBCO. É preciso fazer algum código manualmente e alem disso mover ficheiros de umas localizações para outras, por cada serviço, já que não é importado os *schemas* do canónico mas sim copiados. A curva de aprendizagem é muito maior, mesmo com uma *framework.* Os adaptadores são mais difíceis de configurar e com muito mais *bugs* que os adaptadores da TIBCO. É uma boa ferramenta para integrar com produtos da Oracle.

WSO2 tem como IDE o Eclipse. Apesar de se ter uma ferramenta gráfica, 98% do desenvolvimento, o programador tem que criar código manualmente. O código a ser desenvolvido tem de ser feito em XML, e em mapeamentos é utilizado XPath que tem de ser feito manualmente. É a ferramenta que utilizei com a qual se demora mais a desenvolver um único serviço. O próprio IDE em modo de código contem *bugs* graves, como por exemplo leva muitas vezes código que esteja correto a ser considerado incorreto pelo IDE. O que leva a várias verificações adicionais para validar o código manualmente. As licenças são gratuitas pagando só o suporte que o cliente requerer. Os manuais são incompletos e os próprios tutoriais são desfasados da realidade. A informação para a resolução de problemas não se encontram facilmente na internet, ou a maior parte das vezes são inexistentes. Bom para desenvolver poucos serviços, e que não sejam preciso fazer atualizações aos mesmos frequentemente.

1.8 Plano de Trabalho

5 de setembro a 3 de outubro:

- Análise dos requisitos do banco.

6 de outubro a 31 de dezembro:

- *Technical architectural document*
- Desenho do modelo de base de dados
- Desenho da *Framework*

2 de janeiro a 3 de abril:

- *Scripts* de base de dados
- Desenvolvimento da *Framework*

6 de abril a 20 de março:

- Testes
- Documentação
- Formação

Capítulo 2

Bibliografia

[1] Enterprise Application Integration (EAI) disponível em: http://www.peterindia.net/EAIOverview.html última vez visitada em: 2 de junho de 2015.

[2] Cooperative Bank relatórios anuais disponível em: http://www.co-opbank.co.ke/index.php/about-us/annual-reports última vez visitada em: 2 de junho de 2015.

[3] Service Oriented Architecture: What Is SOA? disponível em: http://www.opengroup.org/soa/source-book/soa/soa.htm última vez visitada em: 4 de junho de 2015.

[4] Chapter 1: Service Oriented Architecture disponível em: https://msdn.microsoft.com/en-us/library/bb972954.aspx última vez visitada em: 4 de junho de 2015

[5] SOA_Principles_Poster disponível em: http://serviceorientation.com/index.php/serviceorientation/index última vez visitada em: 4 de junho de 2015

[6] M. Hadi Valipour; Bavar AmirZafari; Kh. Niki Maleki; Negin Daneshpour (2009). "A brief survey of software architecture concepts and service oriented architecture". *2009 2nd IEEE International Conference on Computer Science and Information Technology*. pp. 34–38. doi:10.1109/ICCSIT.2009.5235004. ISBN 978-1-4244-4519-6.

Disponível em: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5235004 última vez visitada em 6 de junho de 2015.

[7] Tony Shan (2004). "Building a service-oriented e *Banking* platform". *IEEE International Conference on* Services *Computing, 2004. (SCC 2004). Proceedings. 2004*. Páginas 237–244. doi:10.1109/SCC.2004.1358011. ISBN 0-7695-2225-4.2004. Disponível em:

http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1358011 última vez visitada em 6 de junho de 2015.

[8] E. Oliveros (2012). *Web Service Specifications Relevant for Service Oriented Infrastructures*. Achieving Real-Time in Distributed Computing: From Grids to Clouds. Páginas 174–198.

[9] Christopher Koch A New Blueprint For The Enterprise, *CIO Magazine*, March 1, 2005, disponível em: http://www.cio.com.au/article/30960/new_blueprint_enterprise última vez visitada em 9 de junho de 2015.

[10] Enterprise Service Bus: Theory in Practice by David Chappell, 2004, ISBN 0596006756

[11] Design Patterns, disponível em: http://www.vincehuston.org/dp/index.html#objectify última vez visitada em: 10 de junho de 2015.

[12] TIBCO, disponível em: http://www.tibco.com última vez visitada em 11 de junho de 2015.

[13] Application Integration Architecture da Oracle disponível em: http://www.oracle.com/us/products/applications/application-integration-architecture/overview/index.html última vez visitada em 11 de junho de 2015

[14] WSO2 Enterprise Service Bus disponível em: http://wso2.com/products/enterprise-service-bus/ última vez visitada em: 11 de junho de 2015.

[15] TIBCO Architecture Fundamentals, Paul C. Brown, ISBN: 032177261X

[16] Architecting Composite Applications and Services with TIBCO, Paul C. Brown, ISBN: 0321802055

[17] Enterprise Integration Patterns, Gregor Hohpe, ISBN: 0321200683

[18] TIBCO User's Guides, disponível em: https://docs.tibco.com/ última vez visitada em 20 de março de 2015.

[19] SOA Design Patterns, Thomas Erl, Prentice Hall, 2009, ISBN: 0136135161